規則限定Resolutionにより証明可能な命題論理式の複雑さ(アルゴリズムと計算量理論)

AUTHOR(S):

MIYANO, Eiji; IWAMA, Kazuo

数理解析研究所講究録
906巻 1995年 47-54

規則限定 Resolution により証明可能な命題論理式の複雑さ

宮野 英次 (Eiji MIYANO)　　岩間 一雄 (Kazuo IWAMA)
九州大学工学部情報工学科

1. Introduction

Resolution is a proof system for the (coNP-complete) family of unsatisfiable CNF predicates that involves only one rule denoted by $(A+x)(B+\overline{x}) \rightarrow (A+B)$. For example,

$$f = (x_1+x_2+x_3)(\overline{x_1}+\overline{x_2}+\overline{x_3})(x_1+\overline{x_3})(\overline{x_2}+x_3)(\overline{x_1}+x_2)$$

is proved as follows: (i) Merge the 1st and 3rd clauses to get (x_1+x_2). (ii) Merge this (x_1+x_2) and the 5th one to get (x_2). (iii) Merge the 2nd and 3rd clauses to get $(\overline{x_2}+\overline{x_3})$. (iv) Then this is merged with the 4th one to get $(\overline{x_2})$. (v) Now we can get *nil* from (x_2) and $(\overline{x_2})$. Thus f is proved in five steps.

Resolution is complete, i.e., any unsatisfiable predicate can be proved. However, if we take the length of proofs into account, *Resolution* is a relatively weak system. The polynomial unboundedness, namely, the existence of predicates for which super-polynomial proof-steps are needed, was first proved for this *Resolution* [Hak85]. After that the same property has been proved for more powerful systems like depth-k Frege systems [Ajt88, BIK+92]. However, it is still open if the most powerful Frege system, called Extended Frege, is polynomially-bounded (if so, NP = coNP). Several results on the possibility of efficient simulation among proof systems are also known [Kra91, PU92].

Thus, as the power of proof systems increases, the set of predicates provable in polynomial time also increases. This is a sure merit. However, as the power increases, it generally becomes harder to find a proof (a sequence of rule applications as given at the beginning) itself. Since the essential goal of proof systems is to find proofs, that could be an important demerit. This is the reason why a lot of research has been done to "decrease" the power of proof systems by means of, e.g., making the rules simpler and/or imposing several kind of restrictions to nondeterminism [GHRS93]. Actually, proofs can be found in deterministic poly-time for several restricted systems such as Unit-Resolution and Horn-Resolution [MSS90].

In this paper we investigate "the tree-form restriction" which has been popularly used for many proof systems [Kra91, IP94] and impose it to *Resolution*. *Tree-Resolution* is a *Resolution* but each clause can be used at most once in its proof. (In the proof given above, the 3rd clause is used twice.) Then the proof can be drawn as a binary tree instead of a directed acyclic graph in normal *Resolution*. *Tree-Resolution* runs in polynomial time for every predicate, namely the set of provable predicates is now in NP. Not surprisingly, therefore, it is no longer complete; such a simple predicate as above f cannot be proved (see Sec. 2). Thus the restriction seems to be very strong and it is somehow reasonable to expect its benefit, easiness of finding proofs.

Unfortunately, as is shown in this paper, *Tree-Resolution* is still intractable and it constitutes a rich hierarchy in terms of the repeated use of clauses. Let $R(k)$ be the set of predicates that are proved by *Resolution* with at most k repetitions of clauses (see Sec. 2 for details). Namely $R(0)$ is the set of predicates provable by *Tree-Resolution* and the opening example f is in $R(1)$. Our main results include: (i) $R(0)$ is NP-complete, and (ii) $R(k)-R(k-1)$ is $\mathrm{D^P}$-complete for any positive integer k.

We may conclude that the tree-form restriction, although popular in many proof systems, does not pay in the case of *Resolution*; it appears to lose too much power (even $R(1) - R(0)$ is D^{P}-complete) and is still intractable. It is a little surprising that only one more application of the clause repetition increases the set of provable predicates enormously.

2. Propositional Proof Systems

In this paper, only CNF predicates are considered. A *literal* is a logic variable x or its negation $\overline{x}$. A *clause* is a sum of literals like $(x_1 + \overline{x_2} + \overline{x_3})$ but it cannot hold two or more same variables; so those like $(x_1 + x_1 + x_2)$ and $(\overline{x_1} + x_1 + x_2)$ are prohibited. A special clause that consists of 0 clauses is denoted by *nil*. A (CNF) *predicate* is a product of clauses like $f = (x_1 + \overline{x_2} + \overline{x_3})(x_2 + x_3)(\overline{x_1})(x_2 + x_3)$. Sometimes we regard that a predicate is a (multi)set of clauses, i.e., above f is a set of four clauses. (Note that f contains two same clauses, which is allowed.) A specific *assignment* of *true* (or 1) and *false* (or 0) into the variables determines the value (true or false) of the whole predicate, which is calculated in the usual way. For example, above $f = 1$ for assignment $(x_1, x_2, x_3) = (0, 0, 1)$. If an assignment makes the value of some clause false, then it is said that the assignment *is covered by* that clause. A predicate is said to be *satisfiable* (*unsatisfiable*) if there is an (no) assignment that makes the predicate true. In other words, a predicate is unsatisfiable iff every assignment is covered by some clause. As mentioned above, $(0, 0, 1)$ is covered by no clause of f and so f is satisfiable.

Resolution is a proof system to show the unsatisfiability of a given predicate, which can be formulated as a nondeterministic algorithm as follows:

Algorithm *Resolution*

Input: predicate $f = \{C_1, C_2, \cdots, C_n\}$

Step 1. Let $S = f$.

Step 2. Apply either the following (i) or (ii).

(i) Select any clause C in S nondeterministically and replace it by two C's, i.e., $S \leftarrow S \cup \{C\}$. (The variable S holds a multiset.)

(ii) Select nondeterministically two clauses C_i and C_j in S satisfying that there is exactly one variable, say, x, such that C_i contains x and C_j contains $\overline{x}$. Then replace those C_i and C_j by a single clause C_{ij} containing of all the other (possibly zero) literals of C_i than x and all the other (possibly zero) literals of C_j than $\overline{x}$. (If some literal, say, y, appears both in C_i and C_j, y appears in C_{ij} only once.)

Step 3. If S contains *nil* then halt and output "f is unsatisfiable". Otherwise return to Step 2.

A *proof* for a predicate f is a sequence of predicates $S_0 = f, S_1, \cdots, S_i, \cdots, S_m = nil$ where each S_i $(i = 1, \cdots, m)$ is the value of variable S at the end of ith round of Step 2. Note that in (ii) of Step 2, C_i and C_j are removed from S. If we need them later, we should duplicate them by (i) in advance.

Proposition 1[DP60]. Any unsatisfiable predicate can be proved by *Resolution*. Namely *Resolution* is complete.

Proof systems based on axioms and inference rules are generally called *Frege systems*. In this sense, *Resolution* can be regarded as a specific depth-2 Frege system. (Depth-2 means that only CNF formulas are involved.) For comparison, we introduce the most powerful depth-2 Frege system [PU92], denoted by *2Frege*, as the following nondeterministic generator: (*Resolution* can also be changed into the form of generators easily.)

Generator *2Frege*

Step 1. This time, variable S holds a (multi)set of predicates (not clauses as before). Let S be the empty set initially.

Step 2. Apply one of the following rules:

(i) Add predicate $x_0\overline{x_0}$ to S. (We often use this simplified notation instead of $(x_0)(\overline{x_0})$.)

(ii) Select (nondeterministically, the same for below) a predicate f in S and duplicate it.

(iii) Select a predicate f and if f contains two same clauses then delete one of them.

(iv) Select a predicate f and replace it by fA where A may be any clause.

(v) Select a predicate f, a clause in f and a literal in the clause. Then delete that literal.

(vi) Select two predicates f_1 and f_2 that can be written as $f_1 = Af$ and $f_2 = Bf$ (A and B are single clauses. Namely f_1 and f_2 differ in only one clause.) Then replace f_1 and f_2 by single predicate $(A+B)f$ (or by f if A contains x and B contains $\overline{x}$ for some variable x).

(vii) Select two predicates f_1 and f_2 that can be written as $f_1 = (x)f_1'$ and $f_2 = (\overline{x})f_2'$ for some variable x. Then replace f_1 and f_2 by single predicate $f_1'f_2'$.

Step 3. Output any predicate in S and halt, or return to Step 2.

Proposition 2. *2Frege* is complete.

Basic difference between *Resolution* and *2Frege* is that during the course of *Resolution*, only one (unsatisfiable) predicate, g, is involved and a new predicate, g', is obtained by a modification of g. In *2Frege*, many (unsatisfiable) predicates are involved and a new predicate can be derived from two (or one) such predicates. (*Resolution* also involves a lot of *clauses* each of which may be regarded as a predicate. However, such a predicate (= clause) cannot be an unsatisfiable predicate excepting *nil.*) It is not hard to see that *2Frege p-simulates Resolution*, namely, if there is a *Resolution* proof of length t for a predicate f then there is a *2Frege* proof of $poly(t)$ steps for f. The converse is not true [Urq87]. So, *2Frege* is more powerful than *Resolution.* Neither *Resolution* nor *2Frege* is polynomially bounded [Ajt88, BIK+92, Hak85, Urq87], i.e., predicates for which we need exponential steps exist for both proof systems.

Note that there is a duplication rule both in *Resolution* and *2Frege*, (i) of Step 2 and (ii) of Step 2, respectively. If we eliminate this duplication rule, then the proof can be expressed in the form of a binary tree (each leaf is a clause in *Resolution* and $x_0\overline{x_0}$ in *2Frege*). This tree-form restriction is a popular one to make many systems simpler but the power of systems usually decreases. It is known [Kra91] that we need depth-$(k+1)$ Tree-Frege to p-simulate depth-k Frege without the tree-form restriction. Also, *Hajós Calculus* (*HC*) is strictly more powerful than *Tree-HC* [IP94]. Here *HC* is a proof system for non-k-colorable graphs. However, in these cases, the tree-form reduction does not destroy the completeness of proof systems.

In the case of *Resolution*, the tree-form restriction makes it severely less powerful. *Tree-Resolution* is no longer complete. Take a look at the opening example again:

$$f = (x_1 + x_2 + x_3)(\overline{x_1} + \overline{x_2} + \overline{x_3})(x_1 + \overline{x_3})(\overline{x_2} + x_3)(\overline{x_1} + x_2).$$

Tree-Resolution cannot prove this f. (Reason: Each of the eight assignments $(x_1, x_2, x_3) = (0,0,0)$ through $(1,1,1)$ is covered by only one clause. Any application of the rule, say, $(x_1+x_2+x_3)$ and $(x_1+\overline{x_3})$ being replaced by (x_1+x_2), reduces the number of the covered assignments at least one, from three to two in the case of these two clauses). At the same time, the restriction makes the system very simple; *Tree-Resolution* now runs in polynomial time for any input predicate because a single application of the rule decreases the size of S by one.

From these observations, it might be reasonable to conjecture that if a formula can be proved by *Tree-Resolution* (in polynomial time), then one could find its proof in deterministic polynomial time. If this is true, then it could be a good news for theorem proving, because many formulas in practice could be proved by *Tree-Resolution* or they could be transformed to such ones by simply repeating some clauses a few times. In this paper, we prove that this conjecture is probably not true.

3. Main Results

Suppose that $k(n)$ is a function in n and let $R(k(n))$ denote a set of predicates f that can be proved by *Resolution* using (i) of Step 2, often called the *duplication rule*, at most $k(|f|)$ times. Hence $R(0)$ is a set of predicates provable by *Tree-Resolution*. The opening example f is not in $R(0)$ but in $R(1)$. Obviously, if $k(n)$ is a polynomial then $R(k(n))$ is in NP, namely, every predicate in $R(k(n))$ can be proved by *Resolution* in polynomially many steps. It is said that a set, P, is in class $\mathrm{D^P}$ if P can be written as $P = P_1 - P_2$ for some NP sets P_1 and P_2.

In this paper we prove the following two theorems, which shows that (i) even $R(0)$ is intractable, and (ii) $R(k(n))$ constitutes a rich hierarchy:

Theorem 1. $R(0)$ is NP-complete.

Theorem 2. For any positive integer k, $R(k) - R(k-1)$ is $\mathrm{D^P}$-complete.

Strong conjectures are: (i) For any polynomial $k(n)$, $R(k(n))$ is NP-complete and (ii) $R(k(n)) - R(k(n)-1)$ is $\mathrm{D^P}$-complete. It should be noted that $R(2^n) = R(\infty)$, namely, an exponentially many applications of the duplication rule is enough.

4. Proof of Theorem 1

As mentioned in Sec. 2, $R(0)$ is in NP. To prove its NP-hardness, we use a reduction from 3SAT. Namely, we will show that for a given 3SAT predicate f we can construct another predicate g such that g is in $R(0)$ iff f is satisfiable. For better exposition, we describe the reduction using the following example as f of five variables $\alpha_1, \cdots, \alpha_5$. Generalization is straightforward:

$$f = (\overline{\alpha_1} + \overline{\alpha_2} + \alpha_3)(\overline{\alpha_1} + \alpha_2 + \alpha_4)(\alpha_1 + \overline{\alpha_4} + \alpha_5)(\alpha_2 + \alpha_3 + \overline{\alpha_5})$$
$$(\alpha_3 + \alpha_4 + \overline{\alpha_5})(\overline{\alpha_2} + \overline{\alpha_3} + \alpha_5)(\alpha_1 + \overline{\alpha_2} + \alpha_4)(\alpha_2 + \alpha_3 + \alpha_5).$$

The reduced predicate g consists of five groups of clauses, G_1 through G_5, i.e., $g = G_1G_2G_3G_4G_5$. The first group G_1 consists of the following single clause:

$$\mathbf{G_1}: \quad (\overline{x_1} + \overline{x_2} + \overline{x_3} + \overline{x_4} + \overline{x_5} + a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8). \tag{1}$$

Associated with the five variables of f, we prepare x_1 through x_5. a_1 through a_8 correspond to the eight clauses of f.

The second group G_2 consists of the following five (= the number of variables of f) clauses, where σ is a special single variable playing an important role:

$$\mathbf{G_2}: \quad (x_1 + \overline{\sigma})(x_2 + \overline{\sigma})(x_3 + \overline{\sigma})(x_4 + \overline{\sigma})(x_5 + \overline{\sigma}). \tag{2}$$

G_3 consists of the following 24 (= 3× the number of f's clauses) clauses:

$\mathbf{G_3}$: $\alpha_1 : (\overline{x_1}+\overline{a_3})(\overline{x_1}+\overline{a_7})$ (3)
$\overline{\alpha_1} : (\overline{x_1}+\overline{a_1})(\overline{x_1}+\overline{a_2})$ (4)
$\alpha_2 : (\overline{x_2}+\overline{a_2})(\overline{x_2}+\overline{a_4})(\overline{x_2}+\overline{a_8})$ (5)
$\overline{\alpha_2} : (\overline{x_2}+\overline{a_1})(\overline{x_2}+\overline{a_6})(\overline{x_2}+\overline{a_7})$ (6)
$\alpha_3 : (\overline{x_3}+\overline{a_1})(\overline{x_3}+\overline{a_4})(\overline{x_3}+\overline{a_5})(\overline{x_3}+\overline{a_8})$ (7)
$\overline{\alpha_3} : (\overline{x_3}+\overline{a_6})$ (8)
$\alpha_4 : (\overline{x_4}+\overline{a_2})(\overline{x_4}+\overline{a_5})(\overline{x_4}+\overline{a_7})$ (9)
$\overline{\alpha_4} : (\overline{x_4}+\overline{a_3})$ (10)
$\alpha_5 : (\overline{x_5}+\overline{a_3})(\overline{x_5}+\overline{a_6})(\overline{x_5}+\overline{a_8})$ (11)
$\overline{\alpha_5} : (\overline{x_5}+\overline{a_4})(\overline{x_5}+\overline{a_5})$. (12)

Here, the leftmost labels such as "α_1:" are just for readability. The first four clauses in (3) and (4) mean that the variable of f corresponding to x_1 (i.e., α_1) appears in the 3rd, 7th, 1st and 2nd clauses. Note that α_1 appears in the 3rd and 7th clauses as an affirmative literal and therefore label α_1 is used here.

G_4 consists of 10 clauses:

$\mathbf{G_4}$: $\alpha_1 : (U(1)+\sigma+\overline{x_1}+a_3+a_7)$ (13)
$\overline{\alpha_1} : (U(1)+\sigma+\overline{x_1}+a_1+a_2)$ (14)
$\alpha_2 : (U(2)+\sigma+\overline{x_2}+a_2+a_4+a_8)$ (15)
$\overline{\alpha_2} : (U(2)+\sigma+\overline{x_2}+a_1+a_6+a_7)$ (16)
$\alpha_3 : (U(3)+\sigma+\overline{x_3}+a_1+a_4+a_5+a_8)$ (17)
$\overline{\alpha_3} : (U(3)+\sigma+\overline{x_3}+a_6)$ (18)
$\alpha_4 : (U(4)+\sigma+\overline{x_4}+a_2+a_5+a_7)$ (19)
$\overline{\alpha_4} : (U(4)+\sigma+\overline{x_4}+a_3)$ (20)
$\alpha_5 : (U(5)+\sigma+\overline{x_5}+a_3+a_6+a_8)$ (21)
$\overline{\alpha_5} : (U(5)+\sigma+\overline{x_5}+a_4+a_5)$. (22)

Again "α_1:" and so on are just labels. Clauses (13) and (14) mean that α_1 appears in the 3rd and 7th clauses as a fixed polarity and also appears in the 1st and 2nd clauses as the other polarity. $U(1)$ through $U(5)$ are defined as follows: $U(1) = u_1$, $U(2) = \overline{u_1}+u_2$, $U(3) = \overline{u_1}+\overline{u_2}+u_3$, $U(4) = \overline{u_1}+\overline{u_2}+\overline{u_3}+u_4$, $U(5) = \overline{u_1}+\overline{u_2}+\overline{u_3}+\overline{u_4}$, where u_1 through u_4 are new variables. It is important to see that, for any sum F of literals, $(F+U(1))(F+U(2))\cdots(F+U(5))$ can be changed to (F) by *Tree-Resolution*. But the (proper) order of the rule application is unique; namely we have to merge $(F+U(4))$ and $(F+U(5))$ first to get $(F+\overline{u_1}+\overline{u_2}+\overline{u_3})$ and then this must be merged with $(F+U(3))$ to get $(F+\overline{u_1}+\overline{u_2})$ and so on. If, for example, we first merge $(F+U(1))$ and $(F+U(2))$, then we can never imply F.

The last group, G_5, consists of the following five clauses determined only by the number of variables in f:

$$\mathbf{G_5} : (U(1)+\sigma+x_1)(U(2)+\sigma+x_2)(U(3)+\sigma+x_3)(U(4)+\sigma+x_4)(U(5)+\sigma+x_5). \quad (23)$$

Lemma 1. If f is satisfiable then g is in $R(0)$.

Proof. Suppose that the original 3SAT predicate becomes true by assignment, say, $(\alpha_1,\cdots,\alpha_5) = (0,1,0,1,1)$ (in fact this assignment makes f true). Then we use clauses labeled by $\overline{\alpha_1}, \alpha_2, \overline{\alpha_3}, \alpha_4$ and α_5 in G_3. One can see that among those literals we can always choose eight clauses by which the eight literals a_1 through a_8 in G_1 are all deleted. (Suppose that clauses $C_1 = (x_1+x_2+x_3)$ and $C_2 = (x_1+\overline{x_2})$ are replaced by (x_1+x_3). Then we often say that literal x_2 of C_1 is *deleted* by C_2.) For example, $(\overline{x_2}+\overline{a_2})$ in (5) can delete a_2 of (1), by which we mean $\alpha_2 = 1$ makes the f's 2nd clause true. Then we use five clauses in G_2, each of which can delete $\overline{x_1}$ through $\overline{x_5}$ of G_1 but the literal $\overline{\sigma}$ is added. At this moment, there remain clause $(\overline{\sigma}\ )$ that can be regarded as an offspring of G_1, no clauses of G_2, all the clauses labeled by $\alpha_1, \overline{\alpha_2}, \alpha_3, \overline{\alpha_4}$ and $\overline{\alpha_5}$ in G_3 (none of which was used above), and all the clauses in G_4 and G_5. Then we can now use $\alpha_1, \overline{\alpha_2}, \alpha_3, \overline{\alpha_4}$ and $\overline{\alpha_5}$ in G_3 to delete all a_i's in the clauses labeled by the same $\alpha_1, \overline{\alpha_2}, \alpha_3, \overline{\alpha_4}$ and $\overline{\alpha_5}$ in G_4, which produces $(U(1)+\sigma+\overline{x_1}), \cdots$, and $(U(5)+\sigma+\overline{x_5})$.

Notice that this was able to be done only because all the clauses labeled by $\alpha_1, \overline{\alpha_2}, \alpha_3, \overline{\alpha_4}$ and $\overline{\alpha_5}$ in G_3 remained. This is a key point of the proof. The five literals $\overline{x_1}$ through $\overline{x_5}$ are now deleted by using the five clauses of G_5, which implies $(U(1)+\sigma), \cdots$, and $(U(5)+\sigma)$. Now we can get (σ) from these five clauses , and then can get *nil* from (σ) and $(\overline{\sigma})$. □

Now suppose that the original predicate f is unsatisfiable. We wish to show that the reduced predicate g cannot be proved by *Tree-Resolution.* An intuitive observation is that the clauses in G_3 are not enough to delete both (i) all a_i's in (1) and (ii) all a_i's in some five clauses in G_4 that includes $U(1)$ through $U(5)$ completely. The following lemma will be often used.

Lemma 2. Let the current set of clauses be S. Then if S is satisfiable then we can never imply *nil* from S by *Tree-Resolution.*

Now suppose that we are trying to apply the rule to some two clauses, say, to a clause in G_2 and a clause in G_4. (We shall say that the rule is applied to a (G_2, G_4) pair in such an occasion.) There are 15 possibilities from $(G_1, G_1), (G_1, G_2), \cdots$, through (G_5, G_5). However, the rule can actually be applied to a very few of them at the beginning.

Lemma 3. Suppose that $S = g$. Then it is enough to consider the application of the rule to only (G_1, G_2), (G_1, G_3), (G_3, G_4) and (G_4, G_4) pairs.

Then we next consider what will happen if we merge G_4 and G_4 to check the last case in Lemma 3. Again we have to merge $U(4)$ and $U(5)$ first. For example, suppose that we merged (20) and (21) and got

$$(\overline{u_1} + \overline{u_2} + \overline{u_3} + \sigma + \overline{x_4} + \overline{x_5} + a_3 + a_6 + a_8). \tag{24}$$

Then the following observation holds: (i) We still need (19) and (22) to imply *nil.* The reason is that if we remove either, say, (19), then S becomes satisfiable (details are omitted). (ii) We need all clauses (1), (22) and (24) to get *nil* since if (1) is removed then the predicate becomes satisfiable, since if we do not need (24) then the above merge operation was needless and since we showed that (22) is also needed in above (i). Note that literal $\overline{x_5}$ appears three times in (1), (22) and (24). Although we have to delete all these three $\overline{x_5}$'s, there are only two x_5's in (2) and (23). The same argument also holds for $\overline{x_4}$. So, before using (2) and (23), we have to reduce the number of $\overline{x_5}$'s and $\overline{x_4}$'s into at most two. (iii) To do so, we essentially have to merge some two of those three clauses. Let us first consider the merge of (1) and G_4. First of all it is not possible currently. It would become possible by merging (1) and G_2, by which $\overline{\sigma}$ is added to (1). However, when we merge (1) and G_4, some u_i's must be added to (1), which makes the predicate satisfiable. (iv) Therefore, we have to merge (22) and (24). It is again impossible currently, but would become possible only by merging (22) and $(U(4)+\sigma+x_4)$ in G_5, which introduces u_4 into (22). However, that implies such a clause like $(\overline{u_1}+\overline{u_2}+\overline{u_3}+\sigma+\overline{x_5}+\cdots)$ from (20)-(22), which again makes the predicate satisfiable.

Thus we can conclude that the first merge must be applied to (G_1, G_2), (G_1, G_3) or (G_3, G_4). Suppose that we have applied the rule to (G_1, G_3) and (G_3, G_4) several times but no major changes, such as all a_i's in G_4 disappear, have not occurred yet. Then we can claim the same lemma as Lemma 3 for the current predicate S (proof is very similar and is omitted). So, suppose that we apply the rule to (G_1, G_3) several times, and then to (G_1, G_2), say to (1) and $(x_1+\overline{\sigma})$. Then $\overline{x_1}$ in (1) disappears and we can never apply the rule to (G_1, G_3) if it revives $\overline{x_1}$ in (1) (the predicate becomes satisfiable). Hence, without loss of generality, we can assume that the rule should be applied first to (G_1, G_3) as many times as possible and then to (G_1, G_2). By this sequence of applications, we can get

$$(\overline{x} + a + \overline{\sigma}), \tag{25}$$

where $\overline{x}$ is a sum of some (possibly zero) $\overline{x_i}$'s and similarly for a.

Also, the rule is applied to (G_3, G_4) several times, which deletes a_i's of G_4. When all the a_i's in some clause of G_4 are deleted, that clause can only be merged with G_5. If some clause of G_5 has been used in this way, then all the others must be used in the same way. (If we do not use some clause in G_5, namely if we delete it, then the predicate becomes satisfiable.) Thus we get $(U(1)+\sigma)(U(2)+\sigma)\cdots(U(5)+\sigma)$ and then can get (σ).

At this moment, S contains (25), some of G_2, some of G_3, (σ) and some of G_4. Now our only way of continuing *Resolution* is to further remove $\overline{x_i}$ or a_i from (25) or to merge (25) with (σ). Suppose that we do merge (25) and (σ) before all the $\overline{x_i}$'s and a_i's are removed. Then the result would be $(\overline{x}+a)$ and the predicate becomes satisfiable. Thus we can conclude that all the $\overline{x_i}$'s and a_i's in (25) must be removed using G_2 and G_3. However, one can see that this merging procedure is essentially the same as that conducted in Lemma 1, namely, it follows that g is satisfiable. However, this contradicts the assumption. □

5. Proof of Theorem 2

We first show two key lemmas: The first one is on so-called MINIMAL-UNSAT due to [PW88].

Lemma 4. From any CNF predicate f, one can construct another CNF predicate f' such that (i) if f is satisfiable then so is f' and (ii) if f is unsatisfiable then so is f' but it becomes satisfiable if any one clause is removed from f'.

Let $f = C_1C_2\cdots C_n$ be a CNF predicate. Then, for a variable x, $(x \oplus f)$ denotes predicate $(C_1+x)(C_2+x)\cdots(C_n+x)$.

Lemma 5. Let e_1, e_2 and e_3 be CNF predicates such that (i) e_i and e_j $(i \neq j)$ do not share any same variable and (ii) $e_i \notin R(0)$ for all i. Furthermore none of variables x_1, x_2 and x_3 appear in any of e_i's. Then the following predicate is not in $R(2)$:

$$(x_1 \oplus e_1)(x_2 \oplus e_2)(x_3 \oplus e_3)(\overline{x_1}+\overline{x_2}+\overline{x_3}) \tag{26}$$

Lemma 6. If one of e_1, e_2 and e_3 is not $R(0)$, say if $e_1 \notin R(0)$, then (26) $\notin R(0)$. If two of them are not in $R(0)$, then (26) $\notin R(1)$.

Lemma 7. If $e_1 \in R(k_1)$, $e_2 \in R(k_2)$ and $e_3 \in R(k_3)$ for $k_1, k_2, k_3 \geq 0$, then (26) $\in R(k_1+k_2+k_3)$.

We first prove that $R(1) - R(0)$ is D^{P}-complete. The reduction is from SAT-UNSAT [PY84] which is the problem asking, given two CNF predicates f_0 and f_1, whether or not f_1 is satisfiable and f_0 is unsatisfiable. We shall construct a predicate g, from f_0 and f_1, such that g is in $R(1) - R(0)$ iff f_0 is unsatisfiable and f_1 is satisfiable.

We construct three predicates g_1, g_2 and g_3 as follows. (i) g_1 is reduced from f_1 by the method described in the proof of Theorem 1. (ii) g_2 is also reduced from f_1 in the same way but we use completely different variables from g_1. (iii) As for g_3, we once change f_0 into the minimal unsatisfiable f_0' (see Lemma 4) and this f_0' is then transformed to g_3 in the way of Theorem 1. Again we use new variables. Now g is $(x_1 \oplus g_1)(x_2 \oplus g_2)(x_3 \oplus g_3)(\overline{x_1}+\overline{x_2}+\overline{x_3})$. We shall consider four different cases: (i) f_1 is satisfiable and f_0 is not. Then by Theorem 1, g_1 and $g_2 \in R(0)$. Since f_0 is unsatisfiable and f_0' is its minimum-unsat version, all the clauses except one of f_0' can be made true by some assignment. Then one can see, by a careful observation of the proof of Theorem 1, that $g_3 \notin R(0)$ but $g_3 \in R(1)$. (Namely, only one duplicate operation

for some of group G_3 is enough.) Now, by Lemmas 6 and 7, $g \notin R(0)$ but $g \in R(1)$. (ii) f_1 is not satisfiable and f_0 is not either. $g \notin R(2)$ by Lemma 5. (iii) f_1 is sat and f_0 is sat. $g \in R(0)$. (iv) f_1 is not sat and f_0 is sat. $g_1 \notin R(0)$ and $g_2 \notin R(0)$, so $g \notin R(1)$ by Lemma 6. As a result, $g \in R(1) - R(0)$ iff f_1 is satisfiable and f_0 is unsatisfiable, which is what we wanted to show.

Extension to $R(k) - R(k-1)$ is straightforward: Let F be the following fixed predicate.

$$(v_1 + v_2 + v_3)(v_1 + v_2 + \overline{v_3})(v_1 + \overline{v_2} + v_3)(v_1 + \overline{v_2} + \overline{v_3})$$
$$(\overline{v_1} + v_2 + v_3)(\overline{v_1} + v_2 + \overline{v_3})(\overline{v_1} + \overline{v_2} + v_3)(\overline{v_1} + \overline{v_2} + \overline{v_3}). \quad (27)$$

Let G_i, $1 \leq i \leq k-1$, be the predicate reduced from the above F (G_i and G_j, $i \neq j$, share no variables) by the method of Theorem 1. Then one can see easily that $G_i \notin R(0)$ but $G_i \in R(1)$. Now for given f_1 and f_0 as before, set

$$g = (x_1 \oplus g_1)(x_2 \oplus g_2)(x_3 \oplus g_3)(x_4 \oplus G_1) \cdots (x_{k+2} \oplus G_{k-1})(\overline{x_1} + \cdots + \overline{x_{k+2}}).$$

Then we extend Lemmas 5, 6 and 7 appropriately and consider the four cases just as before: (i) $g_3, G_1, \cdots, G_{k-1} \notin R(0)$, so $g \notin R(k-1)$. g_1 and $g_2 \in R(0)$ and $g_3, G_1, \cdots, G_{k-1} \in R(1)$, so $g \in R(k)$. (ii) $g \notin R(k+1)$. (iii) $g \in R(k-1)$. (iv) $g \notin R(k)$. That concludes the proof of Theorem 2. □

References

[Ajt88] M. Ajtai. The complexity of the pigeonhole principle. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pp.346-355, 1988.

[BIK+92] P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák and W. Woods. Exponential lower bounds for the pigeonhole principle. In *Proc. 24th ACM Symposium on Theory of Computing*, pp.200-220, 1992.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. Assoc. Comput. Mach.*, 1, pp.201-215, 1960.

[GHRS93] D. Gabby, C. Hogger, J. Robinson and J. Siekmann (Ed.). *Handbook of Logic in Artificial Intelligence and Logic Programming*, Clarendon Press, 1993.

[Hak85] A. Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39, pp.297-308, 1985.

[IP94] K. Iwama and T. Pitassi. Exponential lower bounds for the tree-like Hajós calculus. *manuscript*, 1994.

[Kra91] J. Krajíček. Lower bounds to the size of constant-depth propositional proofs. *preprint*, 1991.

[MSS90] S. Miyano, S. Shiraishi and T. Shoudai. A list of P-complete problems. *RIFIS Tech. Rep.*, RIFIS-TR-CS-17, Kyushu Univ., 1990.

[PY84] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *J. Computers and System Sciences*, 28, pp.244-259, 1984.

[PW88] C. H. Papadimitriou and D. Wolfe. The complexity of facets resolved. *J. Computers and System Sciences*, 37, pp.2-13, 1988.

[PU92] T. Pitassi and A. Urquhart. The complexity of Hajós calculus. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pp.187-196, 1992.

[Urq87] A. Urquhart. Hard example for resolution. *J. Assoc. Comput. Mach.*, 34, pp.209-219, 1987.